Mobile Manipulation Planning Optimized for GPGPU Voxel-Collision Detection in High Resolution Live 3D-Maps

Andreas Hermann, Research Center for Information Technology (FZI), hermann@fzi.de, Germany
Jörg Bauer, Research Center for Information Technology (FZI), joebauer@fzi.de, Germany
Sebastian Klemm, Research Center for Information Technology (FZI), klemm@fzi.de, Germany
Rüdiger Dillmann, Research Center for Information Technology (FZI), dillmann@fzi.de, Germany
Address: Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany. Phone: +49 721-9654-242, Fax: +49 721-9654-209

Abstract

Based on the highly parallel voxel-collision-checking on GPGPUs from our previous work, we developed a motion planner for mobile manipulation tasks in full 3D-environments. The presented approaches are optimized with regards to live data consideration during planning and execution monitoring. They allow the generation of complex plans for narrow passage problems but also highly efficient planning in open environments.

1 Introduction

1.1 Motivation

Robots that can be used as flexible helpers in industrial or even domestic environments are a highly desired goal in current research. As mobility is a key ability for such robots, "Mobile Manipulation" has been investigated in robotics for more than a decade. Nevertheless there exist only few solutions that are efficient enough to allow on-the-fly replanning in a dynamic environment with the high dimensionality of mobile manipulation problems. Therefore planning is done offline and execution is accompanied by reactive behaviours to compensate for smaller disturbances. Also current approaches often handle planning of the mobile platform and manipulation planning as separate problems, so the latter one can be solved efficiently in 2D by projecting the robot and the obstacles onto the floor-plane. This is counterproductive in narrow environments as it potentially wastes space. Also a robot would never be able to drive partly underneath an obstacle or reach over a table when searching for a suited manipulation pose. Therefore we wanted to do platform planning in full 3D at a speed, that allows replanning without stopping the motion, when new obstacles emerge. Also we wanted to be able to use the same collision models in full resolution for planning and monitoring during execution.

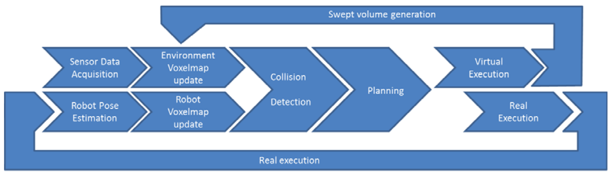

Figure 1: Main program loop showing interleaved planning and execution during continuous data acquisition

1.2 Application scenario

Our robot IMMP (shown on the left in **Fig. 2**) will be deployed in the clean-room-facilities of chip-producing industries. Due to their high maintenance costs the workspace is very limited and stagnancy time due to replanning is expensive. Also the space is shared with human workers. Our motion planner has to deal with these constraints.
Because planning happens in full 3D, the software is also perfectly suited for mobile robots that have a variable footprint due to an articulated upper body. Therefore we also use the planner for our domestic robot HoLLiE from **Fig. 2**.

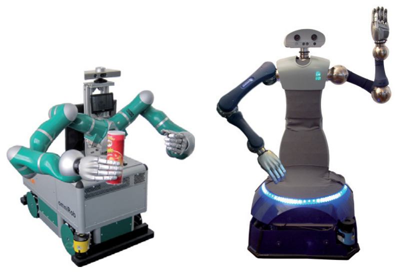

Figure 2: The mobile manipulation platforms IMMP and HoLLiE that utilize the presented motion planner.

1.3 Related Work

Most collision checking software (also the GPU based framework from [1]) follows a mesh-based approach, where all geometries are represented by triangles. This becomes inefficient when collisions between models and live data from 3D-cameras should be detected, as point-clouds have to be triangulated first before intersection tests can be conducted. Therefore we use cell decomposition methods (voxel-maps) for collision checking, which to our best knowledge currently only exist as CPU-implementations (OctoMap [2], Collider).

ISBN 978-3-8007-3601-0

Apart from the mentioned down-projection of robot and environment there also exist approaches that slice the robot model into different levels (platform, body, upper body with arms) which are represented by simplified geometries. This already allows a more detailed planning but fails for flexing body structures, like HoLLiEs. Therefore we take a more general approach and model all geometries with almost no simplification.

The presented planner implements a state of the art sampling based D*-lite planner [3] that reuses valid parts of plans when changes in the environment make replanning necessary. Planning happens in parallel to the execution to achieve a reactive behaviour, like e.g. [4] suggest.

2 Swept-Volume Collision Checks for Motion Primitives

In previous work we demonstrated the capabilities of our extremely parallelized GPGPU collision detection framework [5]. It implements efficient collision checks between a robot model and 3D-pointcloud data to evaluate trajectories virtually during planning but also live during execution. Therefore two equally sized voxel-maps are maintained in GPU RAM: One containing live environment information and the other one containing the robot. For a collision check, the maps are superimposed to find regions which are occupied in both maps. A key feature of the approach is that it allows to not only check single robot poses, but whole trajectories (represented by swept volumes, see **Fig. 3**) within constant time bounds. So even if a single collision check may be slower than in state-of-the-art mesh based libraries, it outperforms other approaches by evaluating many poses or even many swept volumes within one time-cycle. To fully exploit this feature, we implemented a special motion planner, that uses motion primitives to speed up map maintenance and queries many collision checks in parallel.

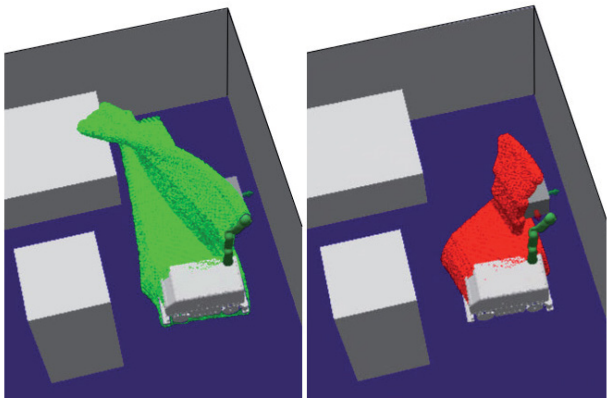

Figure 3: Planning with Swept-volumes. Left: The volume is collision free. Right: Obstacle moved into the trajectory, so the red subvolume lies in collision.

We implemented two processing chains for collision detection:

Map-Map-Querying

When large swept volumes that densely populate a map have to be checked for collisions, this can be achieved efficiently by comparing the whole environment map with the robot planning map. In this case, robot poses are rendered to voxel-addresses according to the results of the direct kinematic and are inserted into the robot map. After that, the maps are superimposed and collisions are searched. This process is repeated for every new pose, until a swept volume is created that can then be monitored during the execution. For details please refer to [5].

Map-List-Querying

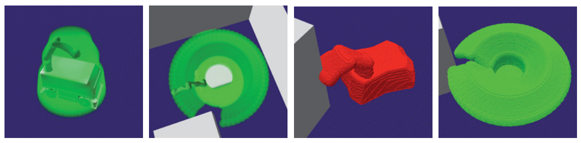

Figure 4: 3D Swept-Volumes of a rotating robot.

For a standalone planning process it is superfluous to check whole maps as only the volume of the map that is covered by the robot has to be investigated in the environment map.

Therefore we don't insert robot poses into a voxel-map but store equivalent voxel-indices of the pose or small Swept-Volumes (**Fig. 3**) in lists. As the addressing scheme is the same as in the maps, we can still query the robot voxels in the environment map for collision checks. This is more efficient, as fewer voxels have to be compared.

More time can be saved when motions are planned, that only move the mobile platform but not the body or the arms. Here the voxelization has to be done only once via the full direct kinematic model and afterwards the resulting voxel-list can be moved within the environment map. The voxel-indices implicitly represent coordinates in the voxel-space, so the position can be altered by adding an offset to all indices in the list. The translation and collision-check of a whole robot-volume is then possible in less than 0.3 ms.

This is only a valid approach for xy-transitions and not for rotations of the cached volume in the list. To handle rotations, instead of caching only a single pose, we cache a rotational swept volume of the robot (see **Fig. 4**). The underlying data-structure makes it possible to keep single rotations distinguishable. That way, a single swept-volume voxel-list covers all rotational angles, and by shifting it inside the environment map, all collision free positions and orientations can be determined.

The inherent discretization of the statespace to the voxel-map resolution of 1 cm^3 is minimal and can be neglected. Also we don't cause discretization errors by rotating the voxel-list, as we are working with the sweep of a rotating robot. This technique is optimally combinable with motion primitives planning and allows fast collision checks without additional memory costs.

3 Planning Algorithm and Path Cost Calculation

In this section we will describe our highly specialized planning approach that is optimized with regards to the voxel-collision-detection system.

3.1 Grid based Swept-Volume Planner

Motion planning for the platform uses a a 8-grid-based graph. Samples represent rotational Swept-Volumes of the robot which consist of 64 subvolumes that can be collision-checked at once with the earlier mentioned algorithm. Collision free subvolumes of different samples are merged into one graph node in the grid, called state. This reduces the graph-size while still allowing resolution complete planning in the whole platform $\mathcal{C}$-Space (x, y, θ). The relationship between the 8-grid, nodes and the graph-states to plan on is shown in **Fig. 5**.

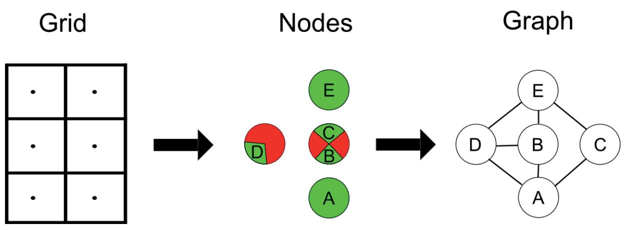

Figure 5: Relation between the grid Cells and the underlying graph. Node C is not connected to node D because their valid orientation range is not overlapping. Therefore no collision free path exists between them in C_{Free}

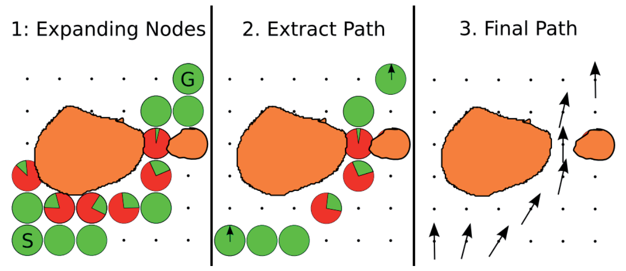

Figure 6: Generation of rotational Swept-Volume-Samples and planning with them on a grid.

Transitions within the grid are possible between neighbours, but are restricted to nodes that share a common collision free orientation (see Fig. **Fig. 6**). On a graph with an appropriate resolution this ensures valid transitions without requiring further interpolating collision-checks.
A desirable property of a platform path is the minimization of rotations, but a cost function for the proposed planner is not straight forward to design, since the states are not representing single configurations of the robot. The chosen implementation makes sure that a transition between two states is only causing additional rotational cost, when a rotation of the platform is actually inevitable.
For this purpose every state, holds an orientation of the platform that is passed on to its successors along the plan when determining motion costs of a plan.
It is checked whether the orientation of the predecessor state is collision free at the successor. If not, an additional cost is added (as the robot has to rotate to take that path) and the closest collision free orientation is saved in the successor.
The orientation saved in the state is not coercively the orientation of the final path at that point as it is smoothed when extracting the path that lies within the orientation range of a node. But this cost function keeps the planner from choosing nodes with a very deviating orientation range. So it enforces paths with few rotations while it still allows complicated manoeuvres with driving the platform sideways and backwards.
Besides the cost for rotation of the platform, the platform should also prefer to move forward, instead of sidewards or even backwards. There may exist situations when it is appropriate or even necessary to move backwards. So one advantage of our planner is to consider such paths. However it is not useful to extend nodes that lie backwards in the driving path if driving forward is also possible. Therefore the planner should extend the backwards nodes only if the forward nodes don't find a path or the path is way longer. This is achieved by a high cost value for backward driving.
The described planner exploits the capabilities of our voxel-collision checker and can therefore efficiently create platform motions for holonomic platforms with variable shapes.

3.1.1 Evaluation of Parameters for the Platform Planner

This section evaluates the planning time with different parameters and the resulting trajectories of the platform planner in three simulated scenarios.
Two factors that greatly affect the total planning time are the grid size and the voxelsize. A large grid-cell size will result in fewer nodes and thus fewer collision checks. But narrow passages may not be found. The voxelsize at which the collision checks are performed is only affecting the time for collision checks. The values for the grid-size have to be multiples of the voxelsize. This is because the collision checks can only be conducted at discrete steps of the voxelsize. Realistic values for the grid size are 0.04 and 0.08 m. Accordingly the collision check can be conducted with voxelsizes of 0.01, 0.02 and 0.04 m.
It is also possible to conduct multiple following collision checks in maps with a different voxelsize. This will be examined in section 3.3.

3.1.2 Scenario 1: Narrow passage

The scenario demonstrates the ability to navigate in narrow passages. **Fig. 7** is showing the environment and the path that the planner has created. A difficulty is the fact that the box on the table may block the robot arm. So the

ISBN 978-3-8007-3601-0

robot has to turn before it drives through the narrow passage. **Table 1** holds the timings for cell sizes of 0.04 m and 0.08 m. With both cell sizes a path has been found. When using the cell size of 0.04 m the number of cells is four times higher. This affects the total planning time in two ways: More collision checks have to be made and the higher number of nodes causes more time for graph creation and search.

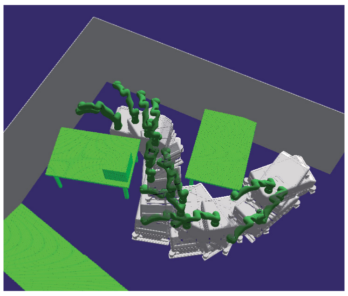
Figure 7: Evaluation scenario 1: The box on the table is on the arms height. Thus the robot has to rotate before it reaches the narrow passage.

Cell Size [m]	Voxel Size [m]	Num. Col. Checks	Col. Check Time [s]	Plan. Time [s]	Path Length [g-val.]
0.04	0.01	2651	2.472	3.621	5.73
0.04	0.02	2634	1.114	2.241	5.77
0.04	0.04	2670	0.546	1.692	5.95
0.08	0.01	745	0.608	0.787	6.17
0.08	0.02	720	0.292	0.471	5.86
0.08	0.04	757	0.139	0.322	5.99

Table 1: The results for the first testing scenario.

The path length is the g-value of the goal node. It consists of the euclidean distances between the grid points and the cost values for rotation and translation. It can bee seen that neither the voxelsize nor the cell size is affecting the path length drastically.

3.1.3 Scenario 2: Smoothness

In the second scenario a longer path through the map is planned. The following **Fig. 8** is showing the smooth resulting path that was generated without additional smoothing steps. The distance to the goal is longer than at the first scenario. But the planning time has not increased significantly. It can be seen that the time for collision checks is even smaller than at the previous scenario although more collision checks have been conducted. The explanation is the fact that the CUDA kernel of the collision check performs additional statistics calculation if a state is in collision. Therefore planning takes longer in narrow passages where many states are in collision. In the current scenario the robot plans through more free space than at the previous. Therefore the collision check is faster.

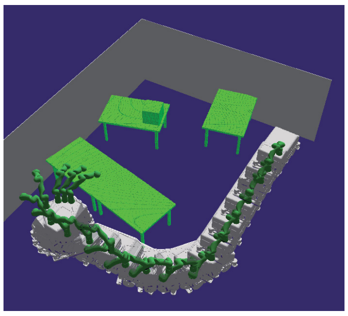
Figure 8: Evaluation scenario 2: Long planning distance. Clean result without additional smoothing.

Cell Size [m]	Voxel Size [m]	Num. Col. Checks	Col. Check Time [s]	Plan. Time [s]	Path Length [g-val.]
0.04	0.01	3412	2.464	4.393	7.54
0.04	0.02	3417	1.060	3.007	7.55
0.04	0.04	3412	0.414	2.331	7.60
0.08	0.01	943	0.628	0.859	7.55
0.08	0.02	1009	0.299	0.560	7.56
0.08	0.04	1214	0.107	0.411	7.70

Table 2: The results for the second testing scenario

3.1.4 Scenario 3: Pointcloud Data

In the last testing scenario a real-world scene has been used. The pointcloud of our lab has been recorded with a rotating laser scanner. **Fig. 9** shows the scenario and the resulting path. A difficulty is the narrow passage. With a grid size of 0.08 m no path could be found. Therefore **Table 3** only shows the results for a grid size of 0.04 m.

Cell Size [m]	Voxel Size [m]	Num. Col. Checks	Col. Check Time [s]	Plan. Time [s]	Path Length [g-val.]
0.04	0.01	1233	1.108	1.688	3.71
0.04	0.02	1208	0.414	1.011	3.72
0.04	0.04	1078	0.161	0.679	3.69

Table 3: The results for the third testing scenario

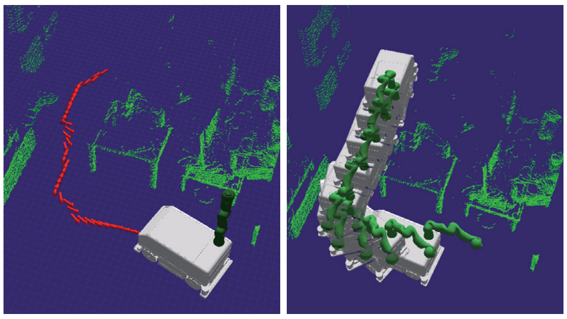
Figure 9: Evaluation scenario 3: Narrow passage planning on a pointcloud of a real-world environment.

ISBN 978-3-8007-3601-0

Summary

The tests have shown that the voxelsize is not very important when planning for the platform. So a voxelsize of 0.04 m should always be chosen for the platform. If a smaller voxelsize is chosen the time for collision checks could be reduced by conducting hierarchical collision checks.
In terms of grid size it has been observed that the most paths could be found with a grid size of 0.08 m. Only if very narrow passages exist a smaller grid size of 0.04 m should be used.

3.2 Replanning

If an obstacle is appearing on the calculated path, it is often not necessary to plan a completely new path because only a small region is blocked. A similar problem is arising when planning is done in an unknown environment. In this case the free space assumption [6] is a common approach to allow planning. It assumes that the unknown part of the environment is not occupied. So a path is created and the robot starts to execute it. During this process the robot will collect new data of the environment and it may turn out that the original path is not feasible. In this case an A* would start planning from scratch. To suppress this, we use an incremental search algorithm that is able to adapt the path to the environment by reusing information about already investigated regions. We use a D*-Lite algorithm, which is easier to implement than the original D* and at least as efficient [7].

Evaluation

In this section the timings for the replanning process will be presented. As mentioned the D*-Lite planner can reuse parts of the plan and create a new plan. To speed up the replanning, the *ComputeShortestPath* function has been altered to run till all nodes of the map have been processed. Then the graph is containing all the information of the static map. So when replanning is performed it won't be necessary to expand new nodes. Different tests have been performed to measure the times till a new plan has been created. Also it has been evaluated how much faster the replanning was compared to a complete new plan at the point where the replanning was performed. Two paths with different length were monitored and an obstacle was put in the way.

Scenario	Col.-Check Time [s]	Planning Time [s]
Scenario 4	0.304	0.605
Replan Scenario 4	0.299	0.567
Scenario 5	0.763	1.511
Replan scenario 5	0.362	0.947

Table 4: The results for the replanning scenarios.

The first replanning-scenario can be seen in **Fig. 10**. In the left image the original path and the new obstacle can be seen. The right image shows the new plan. Whereas the red points are nodes that have gotten inconsistent. The second scenario can be seen in **Fig. 11**.

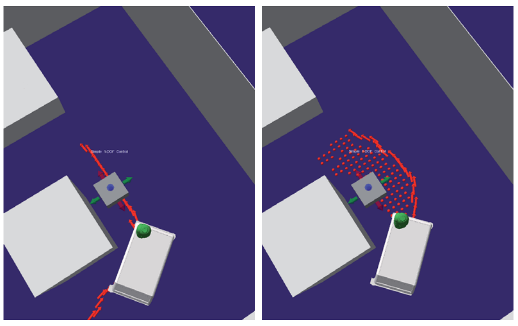

Figure 10: Scenario 4 for replanning. Left: While driving on the red path, a new obstacle is encountered. Right: A new path is planned. The red points show the nodes that became inconsistent.

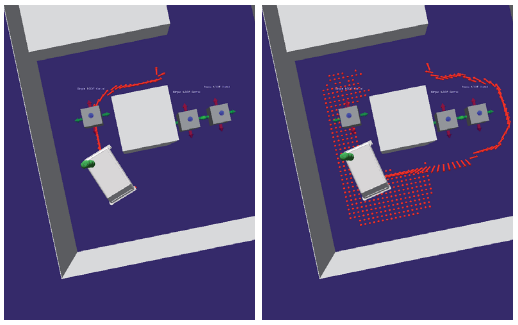

Figure 11: Scenario 5 for replanning. The two obstacles on the right were known. Therefore the platform is choosing the path to the left of the pillar. Then an unknown obstacle is encountered. Again the red points show the inconsistent nodes for which the shortest path to the goal has changed.

Table 4 is holding the results. In the first scenario with a short path the replanning is not much faster. However in the second scenario a reasonable speedup has been achieved. The importance of the feature becomes visible in the real-time experiments with the actual robot, presented in 4.

3.3 Collision Checks in Multiresolution Voxel-Maps

To speed up collision checks in free areas, multiple voxelmaps with different voxel sizes are held in memory. The collision check is first investigating a coarse voxelmap for collisions. Only if one of the states is not valid a second collision check on a finer voxelmap is performed. This method achieves runtimes that are similar to usage of an Octree-data-structure but of course requires more memory on the GPU. New environment data is inserted into the voxelmap with the highest resolution. Then a highly parallelized copy-and-downscale function syncs the information with the different lower resolution maps.

Evaluation

In this section the collision check times for multiresolution voxelmaps will be presented. Also the speedup of the platform planner that is achieved if a hierarchical collision check is used will be evaluated.
Table 5 shows the timings for collision checks in relation to different voxelsizes. Whereby the same collision check has been performed that is used by the platform planner. The speedup to a conventional collision check can be calculated with the following formula.

$$Speedup = \frac{T_{High} * C_{expanded}}{T_{high} * C_{High} + C_{expanded} * T_{Low}}$$

with
T_{high} = average time for high resolution coll. check
T_{low} = average time for low resolution coll. check
$C_{expanded}$ = number of expanded grid cells.
C_{high} = number of necessary high resolution coll. checks

Voxel-size [m]	# Robot Voxels	Coll.-Check Time [ms]
0.01	1889192	0.696
0.02	250818	0.263
0.04	33789	0.089
0.08	4820	0.067

Table 5: Timing for collision Checks

$Voxel\text{-}size_{high}$ [m]	$Voxel\text{-}size_{low}$ [m]	$C_{expanded}$	C_{high}	Speedup
0.01	0.02	839	413	1,149
0.01	0.04	839	434	1,547
0.02	0.04	863	432	1,188

Table 6: Speedup of the collision check when using different combinations of voxel-sizes.

In **Table 6** can be seen that the combination of a voxelsize of 0.01 m and 0.04 m is providing the best speedup. It has to be considered that the collision check is performed with multiple configurations at once. When one of these configurations is in collision the whole collision check is repeated in the high resolution.

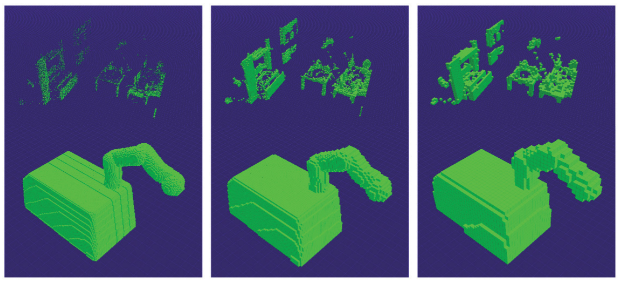

Figure 12: Robot and world representation with voxel-sizes of 0.01, 0.02 and 0.04 m.

3.4 Finding Feasible Manipulation Poses

We took the approach of motion primitive planning one step further and do not only plan with swept volumes of a rotated static robot pose but use sweeps of a motion. This is especially useful to find feasible manipulation poses around a table or a machine. For that, we recorded swept volumes of typical manipulation motions like pick and place, as shown in **Fig. 13** and rotate them around the Tool Center Point of the manipulator. By doing that we can evaluate whole grasp motions in different orientations at varying positions within the environment with only one collision check. The found poses define goals for the platform path planner. **Fig. 14** illustrates the process.

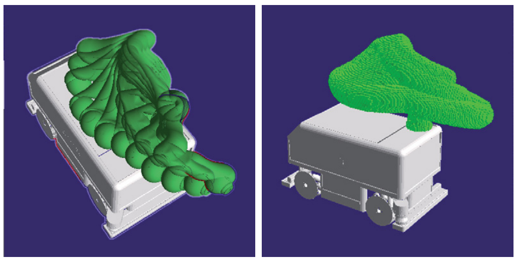

Figure 13: Typical Swept-Volumes of an arm retraction motion and a grasp motion. These volumes can now be rotated around the center of the robot to generate planning states for the evaluation of manipulation poses around a table or a machine.

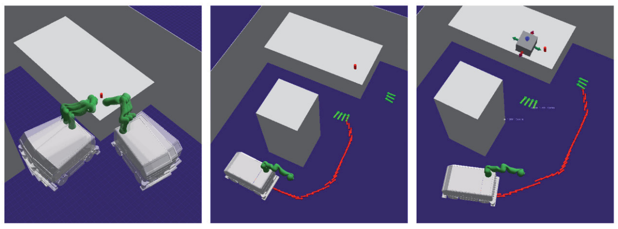

Figure 14: Left: Six valid manipulation poses to reach the red object. Center: Choosing path to closest manipulation pose. Right: Switching to another goal, as a new obstacle on the table is encountered.

3.5 Comparison of an RRT based Planner and our Platform Planner

For a comparison with the state of the art, we tested how fast our platform planner is compared to the *RRTConnect* planner from the OMPL [8]. To conduct collision checks with the pointcloud the OMPL planner will also use the CUDA based collision checking framework. It is only evaluated how the planners perform when planning a platform path. The arm is remaining in a standard configuration.
It was not feasible to adapt the OMPL planner to the new optimized collision check. Therefore it will use much more planning time. To nevertheless compare the results in a meaningful way the number of collision checks will be considered and not the actual planning time. In [9]

ISBN 978-3-8007-3601-0

timings for a state of the art collision checker are published: The time for 1000 mesh based collision checks in a random environment is specified to be 0.240 s. When multiplying the number of collision checks with the timings of the paper, a meaningful comparison can be made.

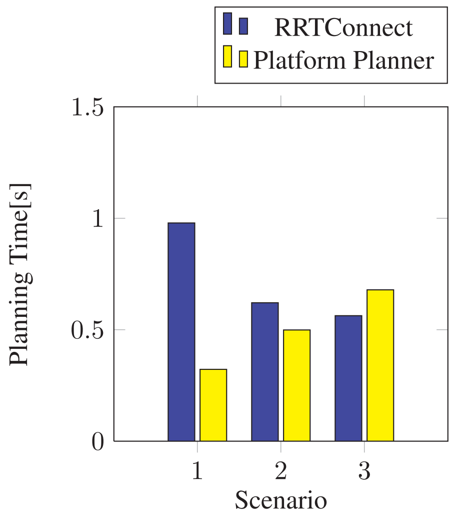

Figure 15: Comparison of the platform planner and the *RRTConnect* planner. The timings for *RRTConnect* are only estimated based on the number of collision checks.

Fig. 15 shows that the platform planner is faster in the two scenarios, where a grid size of 0.08 m has been chosen. It has also to be considered that the estimated time for the *RRTConnect* is only the collision checking time. Whereas the timings for the platform planner are measured over the complete planning cycle.

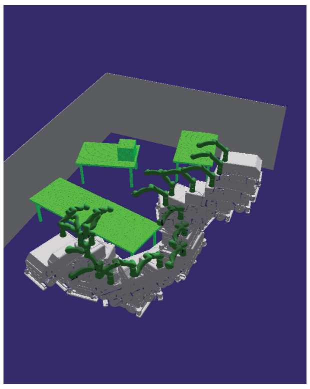

Figure 16: The result of *RRTConnect* for Scenario 2. This can be compared with **Fig. 8** that shows the results of our planner for a similar problem.

An interesting feature of the platform planner is the fact that the planning time is mainly depending on the path length, but not on the actual difficulty of the problem. So the planner is in the first, harder, scenario faster than in the second. Whereas the random sampling based planner needs the most time for the first scenario.
In terms of path quality the platform planner is producing smoother paths than *RRTConnect*. **Fig. 16** shows the resulting path of the *RRTConnect* planner for the second scenario.

4 Experiments

As the onboard GPU on the IMMP robot was not yet ready to use, we tested the real world performance of our algorithms with our robot HoLLiE. Its onboard computer does the controlling but sends the captured 3D data to an off-board GPU where the actual planning is accomplished. The interface between offboard and onboard computers are trajectories of calculated motions which are sent to the robot for execution. We planned in a volume of $10 \times 10 \times 2$ m with a voxel resolution of 0.04 m. Compared to the simulated tests with 1 cm voxel-size this reduces calculation costs about a factor of 64. The robot localization against an independent 2D map is achieved with the laser-range-finders, that allow an accuracy that is sufficient for chosen resolution.
With this parametrization replanning is possible within 0.1 to 0.7 s, depending on where the robot detects a new obstacle. The closer the obstacle is to the current robot position, the more of the already planned path can be reused, which reduces planning time.
When a newly detected obstacle is still in some distance, the robot keeps following his old path while replanning is started. To avoid lags during the switching of trajectories, new trajectories begin in some distance to the robots pose. So the switch-over can happen smoothly. If a new obstacle appears in direct proximity, the robot of course stops for replanning.

5 Conclusions and Future Work

We presented modifications of a state of the art planner to optimally perform with our GPGPU voxel-collision-detection system. Experiments show outstanding planning performance for robots with actuated upper bodies and also the capability of real time monitoring and replanning during the execution. The resulting planning times for full 3D planning are comparable to traditional 2D-projection based planners. This allows reactive behaviours in dynamic environments, represented by point-cloud data. As our algorithms run on a GPGPU also the CPU load is released.
Follow up work will investigate more memory efficient GPGPU Octrees that also allow distance-map calculations for potential-field planning approaches.

References

[1] J. Pan et al., “g-planner: Real-time motion planning and global navigation using GPUs” in *Artificial Intelligence, Twenty-Fourth National Conference on, AAAI 2002*.

[2] A. Hornung et al., “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees” in *Autonomous Robots 2013*.

ISBN 978-3-8007-3601-0

[3] S. Koenig and M. Likhachev, "D*Lite", in *Artificial Intelligence, Eighteenth National Conference on, 2002, pages 476-483*.

[4] I. Reza Nourbakhsh, "Interleaving Planning and Execution for Autonomous Robots", in *Springer International Series in Engineering and Computer Science, 1997*.

[5] A. Hermann et al., "GPU-based real-time collision detection for motion execution in mobile manipulation planning" in *Advanced Robotics, Sixteenth International Conference on, ICAR 2013*.

[6] S. Koenig and Y. Smirnov. "Sensor-based planning with the freespace assumption", *Robotics and Automation, Proceedings. on, International Conference on Robotics and Automation, volume 4, pages 3540-3545. IEEE, 1997*

[7] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain", *Robotics, IEEE Transactions on, 21 (3), pages 354-363, 2005*

[8] I. A. Sucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library", *IEEE Robotics & Automation Magazine, vol. 19, no. 4, pages 72-82, December 2012*, http://ompl.kavrakilab.org.

[9] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries." in: *Robotics and Automation, IEEE International Conference on, ICRA 2012*.

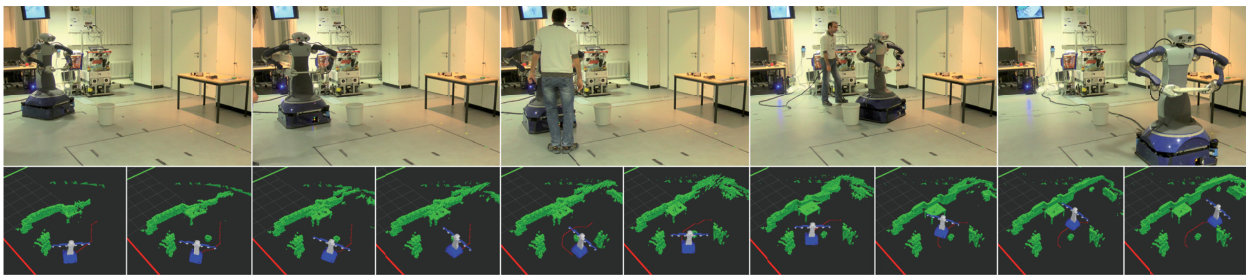

Figure 17: Stills from a live replanning test with the robot HoLLiE. The robot first wants to drive around the waste bin on the right, until a person blocks its path. A new trajectory (red line) on the left is chosen. Replanning happens in 0.1 to 0.7 s, so the robot does not have to stop its motion. The planner uses a voxelmap resolution of 0.04 m voxels.